```java
import java.util.ArrayList;
/**
 *
 * 通过CountDownLatch和CyclicBarrier实现运动员赛跑准备->发令枪开跑->到达终点计算成绩
 *
 * @author renzenggang
 *
 */
public class CountDownLatchTest1 {

    //运动员数量
    private static int SPORTSMAN_COUNT = 10;

    private static final Random random = new Random();

    // 用于判断发令之前运动员是否已经进入准备状态，需要等待10个运动员准备就绪，占有锁，等待10个运动员完成，释放锁。
    private static CountDownLatch readyLatch = new CountDownLatch(SPORTSMAN_COUNT);
    // 用于判断裁判是否已经发令，占有锁，等待裁判发令完成，释放锁
    private static CountDownLatch startLatch = new CountDownLatch(1);
    // 设置终点屏障，用于计算成绩
    private static CyclicBarrier cb = new CyclicBarrier(SPORTSMAN_COUNT, new Runnable() {

        @Override
        public void run() {

            CountDownLatchTest1.transcript
                    .sort((Sportsman p1, Sportsman p2) -> p1.getTranscript() - p2.getTranscript());

            System.out.println("排名成绩单： " + CountDownLatchTest1.transcript);

            CountDownLatchTest1.transcript.clear();
        }
    });

    // 成绩单
    private static List<Sportsman> transcript = new ArrayList<Sportsman>(SPORTSMAN_COUNT);

    public static void main(String[] args) {

        // 用于判断发令之前运动员是否已经进入准备状态，需要等待10个运动员准备就绪，占有锁，等待10个运动员完成，释放锁。
        // CountDownLatch readyLatch = new CountDownLatch(SPORTSMAN_COUNT);
        // 用于判断裁判是否已经发令，占有锁，等待裁判发令完成，释放锁
        // CountDownLatch startLatch = new CountDownLatch(1);

        // 启动10个线程，也就是10个运动员，做准备工作
        for (int i = 0; i < SPORTSMAN_COUNT; i++) {
            Thread t = new Thread(new RunTask((i + 1) + "号运动员", readyLatch, startLatch));
            t.start();
        }
        // 当前运动员在其他运动员准备就绪前一直等待，也就是说等readyLatch倒数计数器为0之前一直等待
        try {
            readyLatch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        // 裁判发令，释放锁
        startLatch.countDown();
        System.out.println("裁判：所有运动员准备完毕，开始跑...");

    }

    // 运动员
    static class Sportsman {
        private String name;
        private int transcript;

        public Sportsman(String name, int transcript) {
            this.name = name;
            this.transcript = transcript;
        }

        @Override
        public boolean equals(Object obj) {
            boolean result = false;
            if (obj instanceof Sportsman) {
                result = ((Sportsman) obj).getTranscript() == this.transcript;
            }
            return result;
        }

        @Override
        public String toString() {
            return this.name + ":" + this.transcript + " ms";
        }

        public String getName() {
            return name;
        }

        public int getTranscript() {
            return transcript;
        }

    }

    // 跑任务
    static class RunTask implements Runnable {

        private Lock lock = new ReentrantLock();

        private CountDownLatch ready;
        private CountDownLatch start;
        private String name;

        /**
         *
         * (构造方法)
         *
         * @param ready
         * @param start
         * @param name  运动员名称
         */
        public RunTask(String name, CountDownLatch ready, CountDownLatch start) {
            this.ready = ready;
```

```java
            this.start = start;
            this.name = name;
        }

        @Override
        public void run() {
            lock.lock();
            try {

                // 1. 写运动员准备就绪的逻辑,准备readyTime秒
                int readyTime = random.nextInt(1000);
                System.out.println(name + ":我需要" + readyTime + "秒的时间准备。");
                try {
                    Thread.sleep(readyTime);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println(name + "我已经准备完毕!");
                // 释放锁readyLatch-1，表示一个运动员已经就绪
                ready.countDown();
                try {
                    // 等待裁判发开始命令
                    start.await();
                } catch (InterruptedException e) {

                }
                System.out.println(name + ": 开跑...");
                int costTime = random.nextInt(500);
                try {
                    Thread.sleep(costTime);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println(name + ": 开跑到达终点。成绩:" + costTime + "ms");
                transcript.add(new Sportsman(name, costTime));
                // 等待成绩
                cb.await();
            } catch (Exception e) {

            } finally {
                lock.unlock();
            }

        }

    }

}
```